Dependency grammars

Formele en natuurlijke talen

Lecture 14

Coming days

Today (dependency grammars)

Tuesday (Probabilistic CFGs)

Thursday (28 March) (Q&A)

Final: Tuesday 2 April, 17.00 - 20.00: Educatorium Gamma (Ruppert)

Goals for Today

Probe a part of language that CFG doesn't capture–dependencies between words

Explain the differences between dependency and constituency grammars

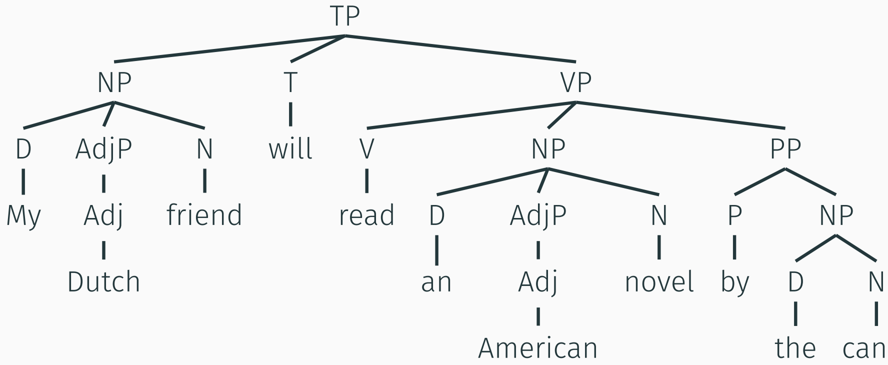
Convert between dependency and constituency representations of the same sentences

Sentence structure

My Dutch friend will read an American novel by the canal.

Sentence structure

My Dutch friend will read an American novel by the canal.

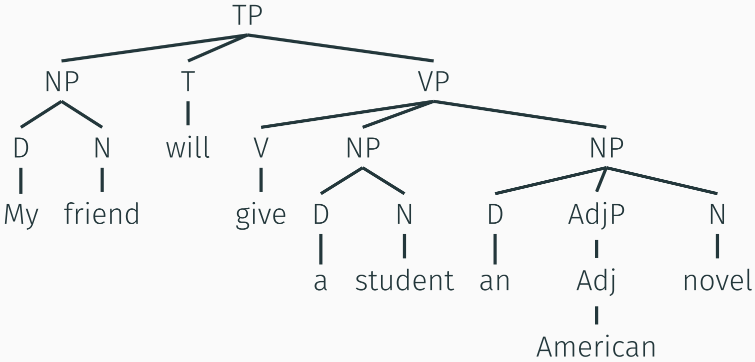


Limits of context-free grammars

Important semantic relations are present in structure but **not directly visible**

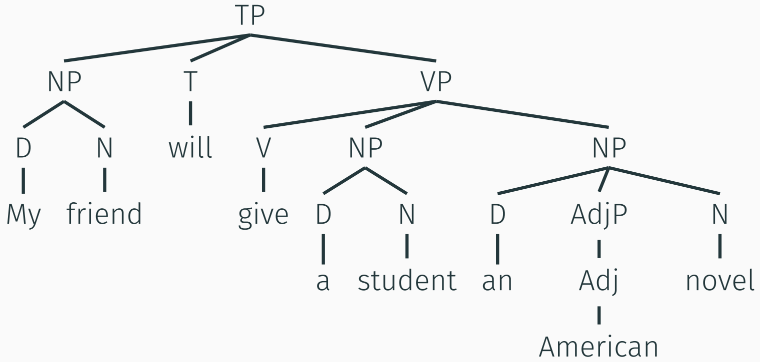
Limits of context-free grammars

Important semantic relations are present in structure but **not directly visible**



Limits of context-free grammars

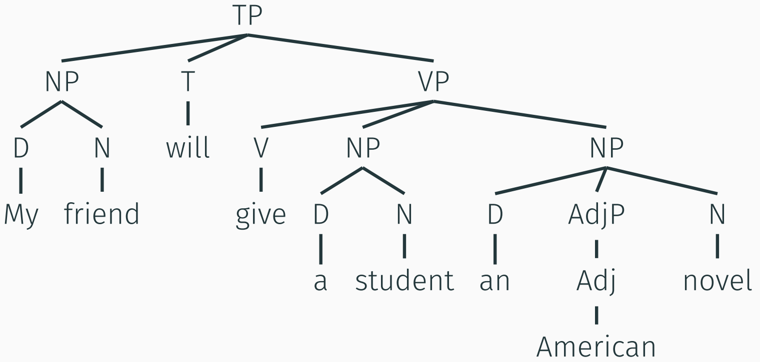
Important semantic relations are present in structure but **not directly visible**



What is the subject?

Limits of context-free grammars

Important semantic relations are present in structure but **not directly visible**

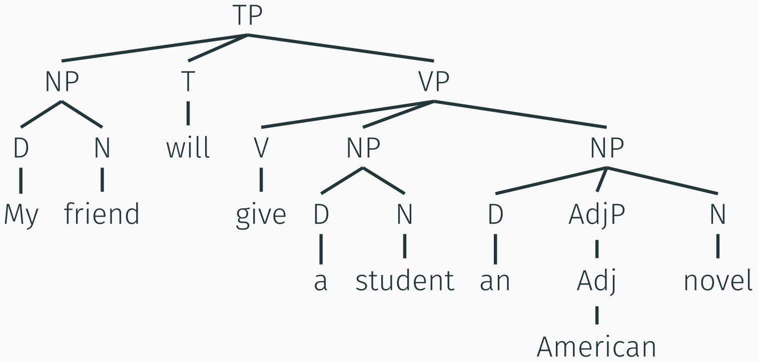


What is the subject?

What is the direct object?

Limits of context-free grammars

Important semantic relations are present in structure but **not directly visible**



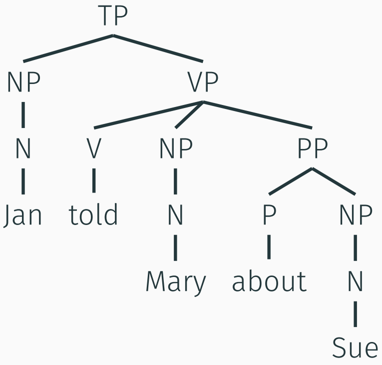
What is the subject?

What is the direct object?

What is the indirect object?

Limits of context-free grammars

Jan told Mary about Sue.



Limits of context-free grammars

Free word order: many options for ways to arrange words.

Limits of context-free grammars

Free word order: many options for ways to arrange words.

Example: Czech

(1) Jan řekl Marii o Zuzaně.
Jan told Marie about Zuzana.
'Jan told Marie about Zuzana.'

Limits of context-free grammars

Free word order: many options for ways to arrange words.

Example: Czech

(1) Jan řekl Marii o Zuzaně.
Jan told Marie about Zuzana.
'Jan told Marie about Zuzana.'

Marie vertelde Jan over Zuzana.

Over Zuzana vertelde Jan Marie.

Jan Marie vertelde over Zuzana.

Jan Marie over Zuzana vertelde.

Marie Jan vertelde over Zuzana.

Marie over Zuzana Jan vertelde.

...

Question 1

Why might a free word order language like Czech pose a problem for context-free grammars?

1) Many production rules are needed to generate every possible word order.
2) Many symbols in the alphabet are needed to generate every possible word order.
3) We cannot capture multiple possible word orders using a context-free grammar.
4) Strings generated with a CFG with multiple word orders cannot be parsed.

Question 1

Why might a free word order language like Czech pose a problem for context-free grammars?

1) **Many production rules are needed to generate every possible word order.**
2) Many symbols in the alphabet are needed to generate every possible word order.
3) We cannot capture multiple possible word orders using a context-free grammar.
4) Strings generated with a CFG with multiple word orders cannot be parsed.

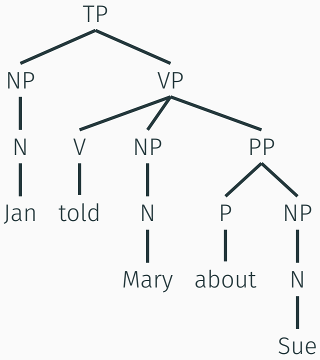
Limits of context-free grammars

A CFG would need additional rules to capture each word order!

Limits of context-free grammars

A CFG would need additional rules to capture each word order!

Jan vertelde Marie over Zuzana.

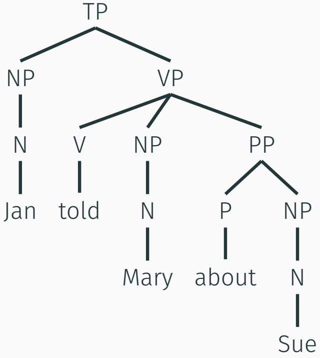


VP $\rightarrow$ V NP PP

Limits of context-free grammars

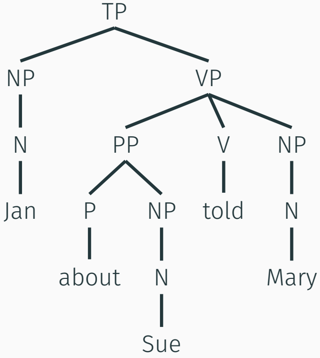
A CFG would need additional rules to capture each word order!

Jan vertelde Marie over Zuzana.



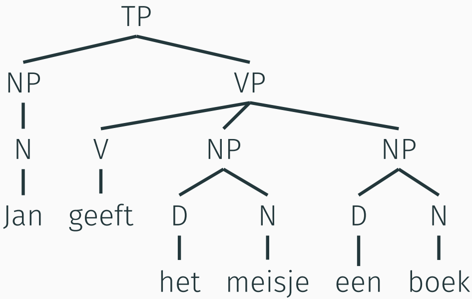
$VP \rightarrow V\ NP\ PP$

Jan over Zuzana vertelde Marie.



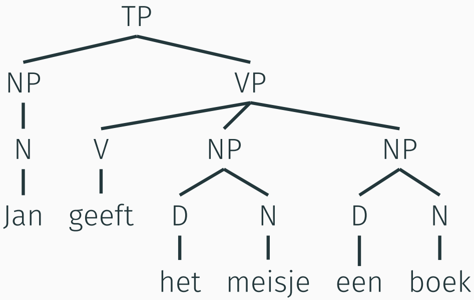
$VP \rightarrow PP\ V\ NP$

Constituents and heads



Context-free grammars indicate which **constituents** are in a sentence.

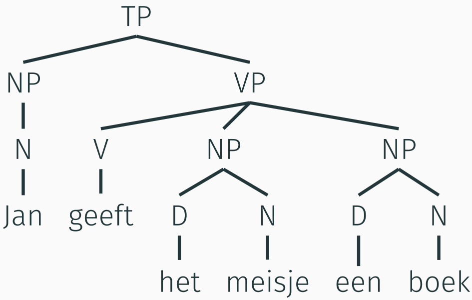
Constituents and heads



Context-free grammars indicate which **constituents** are in a sentence.

Within a constituent, one word is the most important: the **head**

Constituents and heads



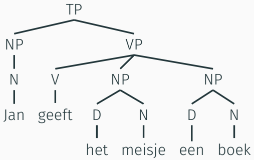
Context-free grammars indicate which **constituents** are in a sentence.

Within a constituent, one word is the most important: the **head**

Other elements: **dependent** on the head

Dependency grammars

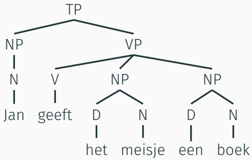
Context-free grammar



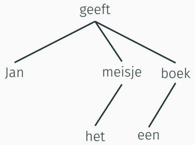
Dependency grammar

Dependency grammars

Context-free grammar

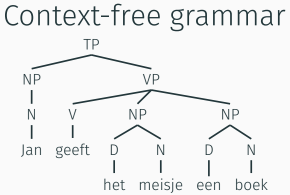


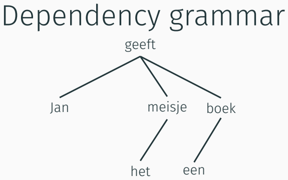
Dependency grammar



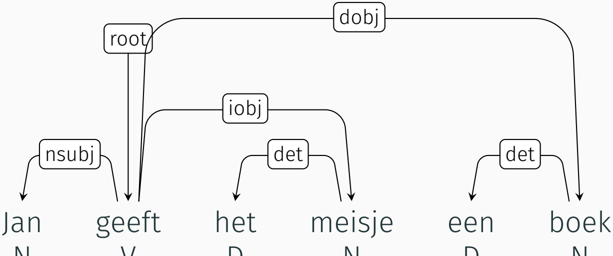
Dependency grammars tell you what **head-dependency** relations are in a sentence.

Dependency grammars





Dependency grammars tell you what **head-dependency** relations are in a sentence.



Dependency trees as graphs

A **dependency tree** is a 3-tuple $\langle V, E, r \rangle$ along with a labeling l.

Dependency trees as graphs

A **dependency tree** is a 3-tuple $\langle V, E, r \rangle$ along with a labeling l.

- V: the set of **vertices** (=plural of vertex)
- $r \in V$: the **root** of the tree
- $E \subseteq V \times V$ the set of (directed) **edges**

Dependency trees as graphs

A **dependency tree** is a 3-tuple $\langle V, E, r \rangle$ along with a labeling l.

- V: the set of **vertices** (=plural of vertex)
- $r \in V$: the **root** of the tree
- $E \subseteq V \times V$ the set of (directed) **edges**
- $l : V \cup E \to L$ for a set of labels L
    - Purpose: identify vertices with words and edges with specific relations (subject, modifier, direct object, etc.)

Dependency trees as graphs

A **dependency tree** is a 3-tuple $\langle V, E, r \rangle$ along with a labeling l.

- V: the set of **vertices** (=plural of vertex)
- $r \in V$: the **root** of the tree
- $E \subseteq V \times V$ the set of (directed) **edges**
- $l : V \cup E \to L$ for a set of labels L
    - Purpose: identify vertices with words and edges with specific relations (subject, modifier, direct object, etc.)

Requirements:

- r has no incoming edges
- Every other vertex has exactly one incoming edge
- Every vertex has a path back to r

Dependency: An example

De jongen zal met een nieuwe pen een brief schrijven.

Dependency: An example

De jongen zal met een nieuwe pen een brief schrijven.

$$\left\langle \begin{array}{c} \{1, 2, 3, 4, 5, 6, 7, 8, 9, \mathrm{r}\}, \\ \left\{ \begin{array}{l} (\mathrm{r}, 2), (\mathrm{r}, 3), (\mathrm{r}, 7), (\mathrm{r}, 9), \\ (2,1), (7, 4), (7, 5), (7, 6), (9, 8) \end{array} \right\} \end{array} \right\rangle$$

$$l = \left\{ \begin{array}{l} (\mathrm{r}, \text{schrijven}), (1, \text{de}), (2, \text{jongen}), (3, \text{zal}), (4, \text{met}), \\ (5, \text{een}), (6, \text{nieuwe}), (7, \text{pen}), (8, \text{een}), (9, \text{brief}), \\ ((\mathrm{r}, 2), \text{nsubj}), ((\mathrm{r}, 3), \text{aux}), ((\mathrm{r}, 7), \text{nmod}), \\ ((\mathrm{r}, 9), \text{dobj}), ((2,1), \text{det}), ((7, 4), \text{case}), \\ ((7, 5), \text{det}), ((7, 6), \text{amod}), ((9, 8), \text{det}) \end{array} \right\}$$

From constituency tree to dependency tree

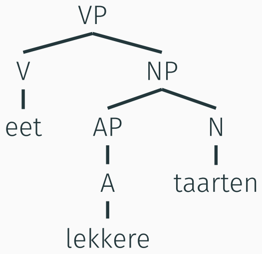
Begin with start symbol C.
Repeat:

- The direct descendant of C consists of a head + dependents. Identify head H of C.
- All other elements are dependents of H.
- For each dependent: identify *its* head H'. Connect H with H'. Its dependent is the new constituent C.

From constituency tree to dependency tree

Begin with start symbol **C**.
Repeat:

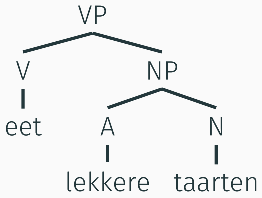
- The direct descendant of **C** consists of a head + dependents. Identify head **H** of **C**.
- All other elements are dependents of **H**.
- For each dependent: identify *its* head **H'**. Connect **H** with **H'**. Its dependent is the new constituent **C**.



From constituency tree to dependency tree

Begin with start symbol **C**.
Repeat:

- The direct descendant of **C** consists of a head + dependents. Identify head **H** of **C**.
- All other elements are dependents of **H**.
- For each dependent: identify *its* head **H'**. Connect **H** with **H'**. Its dependent is the new constituent **C**.



From constituency tree to dependency tree

Begin with start symbol C.
Repeat:

- The direct descendant of C consists of a head + dependents. Identify head H of C.
- All other elements are dependents of H.
- For each dependent: identify *its* head H'. Connect H with H'. Its dependent is the new constituent C.

A dependency tree corresponds with an equivalence class of constituency trees.

From dependency tree to constituency tree

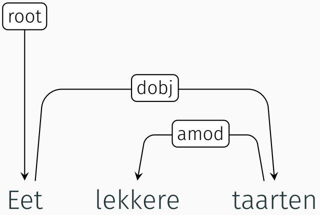
1. Begin with highest element αH of category α (first: the root vertex).
2. Make a node αP.

From dependency tree to constituency tree

1. Begin with highest element αH of category α (first: the root vertex).
2. Make a node αP.

Repeat:

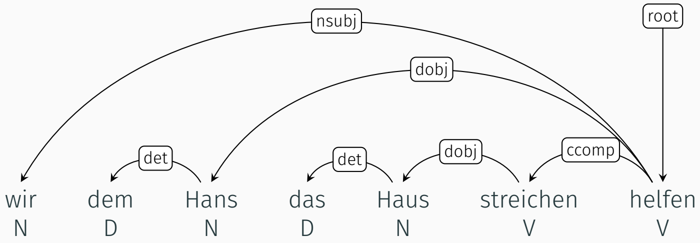
- Make H a daughter of αP.
- For each dependent of H: make a vertex βP (β is the category of the dependent). Connect βP with αP. The dependent is the head of βP.



Projective and non-projective dependency structures

Ich weiss...

...dass wir dem Hans das Haus streichen helfen.

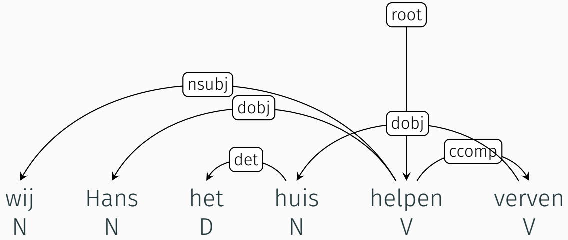


Contiguity between *streichen* ‘paint’ and its direct object *Haus* ‘house’

Projective and non-projective dependency structures

Ik weet...

...dat wij Hans het huis helpen verven

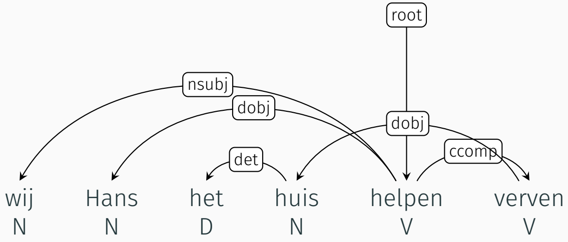


Overlapping dependencies: *verven-huis* and *wij/Hans-helpen*

Projective and non-projective dependency structures

Ik weet...

...dat wij Hans het huis helpen verven



Overlapping dependencies: *verven-huis* and *wij/Hans-helpen*

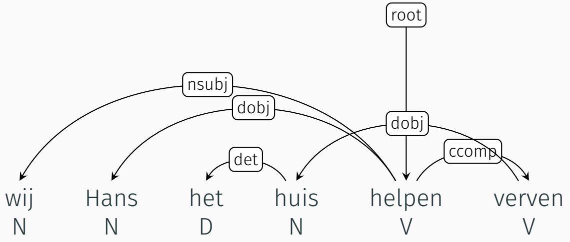
This is a **non-projective** structure.

An arc (connection) between **H** and **D** is projective iff for each word between **H** and **D**, we can find a path starting from H that reaches that word.

Projective and non-projective dependency structures

Ik weet...

...dat wij Hans het huis helpen verven



Overlapping dependencies: *verven-huis* and *wij/Hans-helpen*

This is a **non-projective** structure.

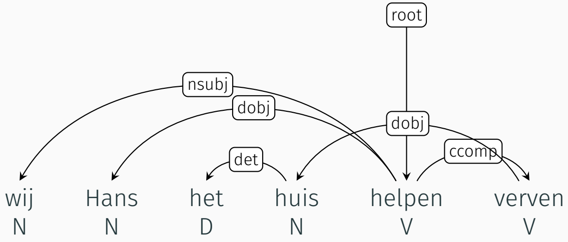
An arc (connection) between **H** and **D** is projective iff for each word between **H** and **D**, we can find a path starting from H that reaches that word.

- Paths must follow the direction of the arrows!

Projective and non-projective dependency structures

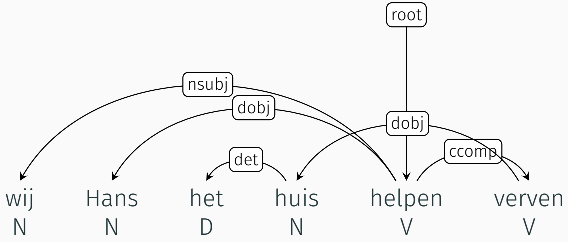
Ik weet...

...dat wij Hans het huis helpen verven



Projective and non-projective dependency structures

Ik weet...
...dat wij Hans het huis helpen verven



Non-projective structures no corresponding structure in constituency trees.

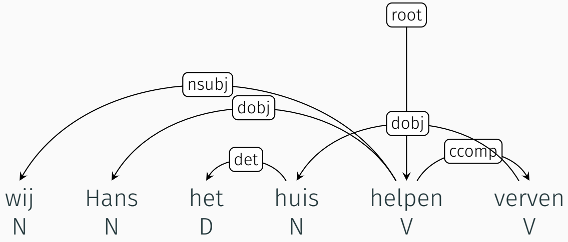
Non-projective structures *also* have no corresponding structure generated by a **CFG**.

Intuition: non-projectivity = arcs that cross one another ('crossing dependencies')

Projective and non-projective dependency structures

Ik weet...

...dat wij Hans het huis helpen verven



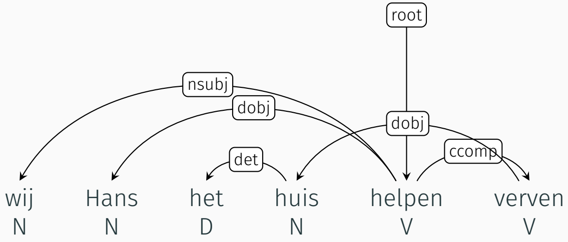
Non-projective structures no corresponding structure in constituency trees.

Non-projective structures *also* have no corresponding structure generated by a **CFG**.

Intuition: non-projectivity = arcs that cross one another ('crossing dependencies')

Projective and non-projective dependency structures

Ik weet...
...dat wij Hans het huis helpen verven



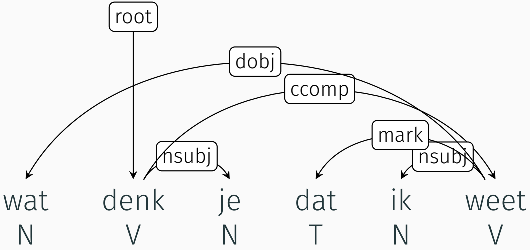
Non-projective structures no corresponding structure in constituency trees.

Non-projective structures *also* have no corresponding structure generated by a **CFG**.

<u>Intuition</u>: non-projectivity = arcs that cross one another ('crossing dependencies')

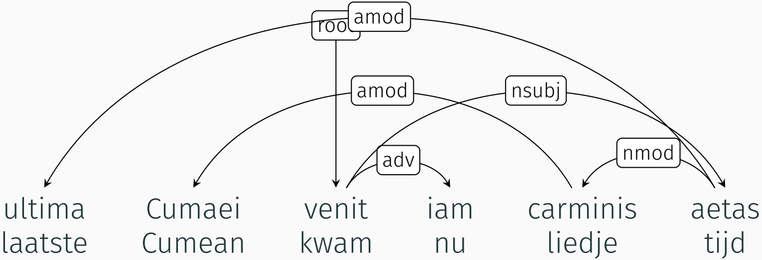
Question 2

Is this dependency tree projective or non-projective?



Question 3

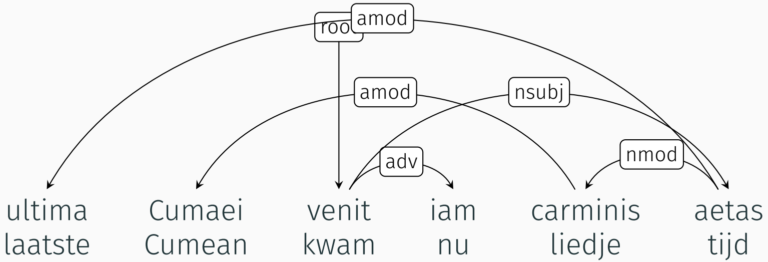
Which of the following dependencies are non-projective?



1) aetas - ultima (amod)
2) venit - aetas (nsubj)
3) carminis - Cumaei (amod)
4) aetas - carminis (nmod)

Question 3

Which of the following dependencies are non-projective?



1) aetas - ultima (amod)

2) venit - aetas (nsubj)

3) carminis - Cumaei (amod)

4) aetas - carminis (nmod)

Non-projective structures

What non-projective structures are part of natural languages?

Non-projective structures

What non-projective structures are part of natural languages?

Jij denkt dat ik een boek zou verbieden.

Non-projective structures

What non-projective structures are part of natural languages?

Jij denkt dat ik een boek zou verbieden.

Welk boek denk je dat ik zou verbieden? (non-proj, ✓)

Non-projective structures

What non-projective structures are part of natural languages?

Jij denkt dat ik een boek zou verbieden.

Welk boek denk je dat ik zou verbieden? (non-proj, ✓)

Welk boek denk je dat mijn beste vriend zou verbieden? (non-proj, ✓)

$\star$The **number of words** in a non-projective path don't play a critical role in determining the acceptability that relation

Non-projective structures

What non-projective structures are part of natural languages?

Of course, some non-projective structures *are* impossible.

Jan is gelukkig omdat hij een interessant boek gelezen heeft.

*Welk boek is Jan gelukkig omdat hij gelezen heeft? (non-proj, *****)

Non-projective structures

What non-projective structures are part of natural languages?

One more impossible case:

Jan kent de vrouw die een boek geschreven heeft.

*Welk boek kent Jan de vrouw die geschreven heeft? (non-proj, *****)

What *does* decide whether a non-projective arc is acceptable?

The **dependency relation between the words inside the arc**

Non-projective structures

What non-projective structures are part of natural languages?

One more impossible case:

Jan kent de vrouw die een boek geschreven heeft.

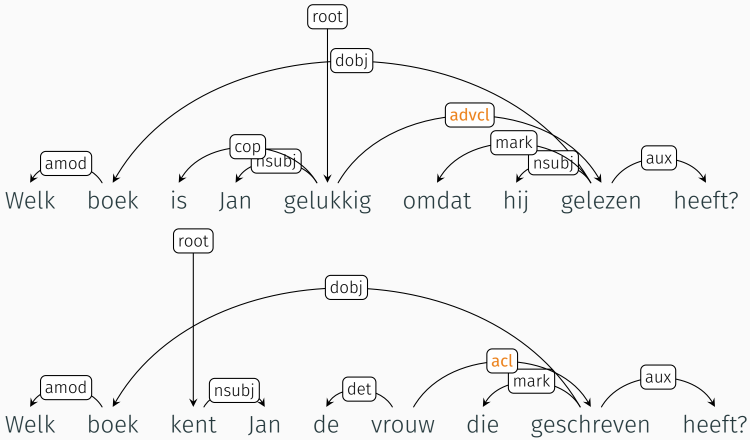
*Welk boek kent Jan de vrouw die geschreven heeft? (non-proj, *****)

What *does* decide whether a non-projective arc is acceptable?

The **dependency relation between the words inside the arc**

Non-projective structures

What non-projective structures are part of natural languages?



Parsing in dependency grammars

Dependency parsing in a nutshell

Like constituency parsing, **derives** a string given a grammar

Dependency parsing in a nutshell

Like constituency parsing, **derives** a string given a grammar

Transition-based parse: Read a string L-to-R, and for each word, assign it as the head of a dependent of some previous word (or vice versa)

- Can be done immediately after reading the word, or word can be stored to do this later

Dependency parsing in a nutshell

Like constituency parsing, **derives** a string given a grammar

Transition-based parse: Read a string L-to-R, and for each word, assign it as the head of a dependent of some previous word (or vice versa)

- Can be done immediately after reading the word, or word can be stored to do this later

Requires consulting an **oracle** (trained independently) to select the right operation to choose

Transition-based dependency parsing

1. Begin with a stack with ROOT.

Transition-based dependency parsing

1. Begin with a stack with ROOT.
2. Do one of the following:
   a) **Left arc**: If α is on top of the stack followed by β, and α is the head and β is the dependent, record the relation and remove β.
   b) **Right arc**: If β is on top of the stack followed by α, and α is the head and β is the dependent, and β has no other dependents in the sentence, record the relation and remove β.

Transition-based dependency parsing

1. Begin with a stack with ROOT.
2. Do one of the following:

   a) **Left arc**: If α is on top of the stack followed by β, and α is the head and β is the dependent, record the relation and remove β.

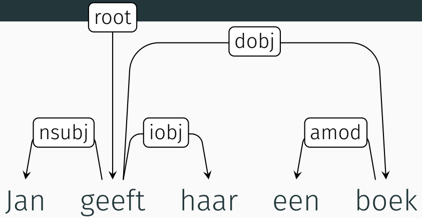
   b) **Right arc**: If β is on top of the stack followed by α, and α is the head and β is the dependent, and β has no other dependents in the sentence, record the relation and remove β.

   c) **Shift**: Take a word from the input and put it on top of the stack.

Transition-based dependency parsing

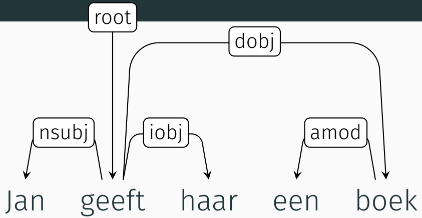
1. Begin with a stack with ROOT.
2. Do one of the following:
    a) **Left arc**: If α is on top of the stack followed by β, and α is the head and β is the dependent, record the relation and remove β.
    b) **Right arc**: If β is on top of the stack followed by α, and α is the head and β is the dependent, and β has no other dependents in the sentence, record the relation and remove β.
    c) **Shift**: Take a word from the input and put it on top of the stack.
3. Repeat step 2 until the input is fully read and the stack consists only of ROOT.

Parsing in dependency grammars



Step	Stack	Word list	Operation	Relation
0	[ROOT]	[Jan, geeft, haar, een, boek]	SHIFT	
1	[ROOT, Jan]	[geeft, haar, een, boek]	SHIFT	
2	[ROOT, Jan, geeft]	[haar, een, boek]	LEFTARC	Jan←geeft
3	[ROOT, geeft]	[haar, een, boek]	SHIFT	
4	[ROOT, geeft, haar]	[een, boek]	RIGHTARC	geeft→haar
5	[ROOT, geeft]	[een, boek]	SHIFT	
6	[ROOT, geeft, een]	[boek]	SHIFT	
7	[ROOT, geeft, een, boek]	[ ]	LEFTARC	een←boek
8	[ROOT, geeft, boek]	[ ]	RIGHTARC	geeft→boek
9	[ROOT, geeft]	[ ]	RIGHTARC	ROOT→geeft
10	[ROOT]	[ ]	RIGHTARC	

Parsing in dependency grammars



Step	Stack	Word list	Operation	Relation
0	[ROOT]	[Jan, geeft, haar, een, boek]	SHIFT	
1	[ROOT, Jan]	[geeft, haar, een, boek]	SHIFT	
2	[ROOT, Jan, geeft]	[haar, een, boek]	LEFTARC	Jan←geeft
3	[ROOT, geeft]	[haar, een, boek]	SHIFT	
4	[ROOT, geeft, haar]	[een, boek]	RIGHTARC	geeft→haar
5	[ROOT, geeft]	[een, boek]	SHIFT	
6	[ROOT, geeft, een]	[boek]	SHIFT	
7	[ROOT, geeft, een, boek]	[ ]	LEFTARC	een←boek
8	[ROOT, geeft, boek]	[ ]	RIGHTARC	geeft→boek
9	[ROOT, geeft]	[ ]	RIGHTARC	ROOT→geeft
10	[ROOT]	[ ]	RIGHTARC	

NB! This parsing method **cannot** be used to parse non-projective structures.

Comparing parsing methods

Why are there so many parsing methods?!

Comparing parsing methods

Why are there so many parsing methods?!

- Parsing methods need to be tailored to the specific kind of grammar (dependency, CFG).

Comparing parsing methods

Why are there so many parsing methods?!

- Parsing methods need to be tailored to the specific kind of grammar (dependency, CFG).
- They also differ in various properties, such as speed, which may depend on properties of the input

Efficiency of parsing methods

Recall: different parsing algorithms take different amounts of time (measured in terms of number of steps they take)

Efficiency of parsing methods

Recall: different parsing algorithms take different amounts of time (measured in terms of number of steps they take)

Example: How many steps for dependency parsing?

2 per word: 1 to shift it onto the stack, 1 to draw its dependency

Efficiency of parsing methods

Recall: different parsing algorithms take different amounts of time (measured in terms of number of steps they take)

Example: How many steps for dependency parsing?

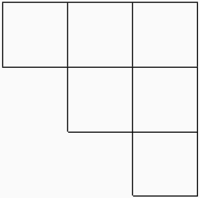
2 per word: 1 to shift it onto the stack, 1 to draw its dependency

Jan geeft haar een boek. $\rightsquigarrow$ 10 steps

Efficiency of parsing methods

Recall: different parsing algorithms take different amounts of time (measured in terms of number of steps they take)

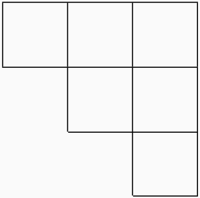
Example 2: How many steps does CYK take?



Efficiency of parsing methods

Recall: different parsing algorithms take different amounts of time (measured in terms of number of steps they take)

Example 2: How many steps does CYK take?

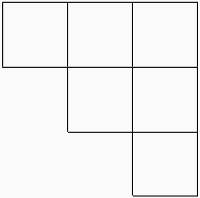


- Need to fill in $n + (n-1) + (n-2) + ... + 1 = \frac{n^2}{2} + \frac{n}{2}$ cells

Efficiency of parsing methods

Recall: different parsing algorithms take different amounts of time (measured in terms of number of steps they take)

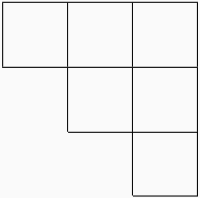
Example 2: How many steps does CYK take?



- Need to fill in $n + (n-1) + (n-2) + ... + 1 = \frac{n^2}{2} + \frac{n}{2}$ cells
- For substrings of length i: need to consider $i - 1$ possible combinations of other cells

Recall: different parsing algorithms take different amounts of time (measured in terms of number of steps they take)

Example 2: How many steps does CYK take?



- Need to fill in $n + (n-1) + (n-2) + ... + 1 = \frac{n^2}{2} + \frac{n}{2}$ cells
- For substrings of length i: need to consider $i - 1$ possible combinations of other cells

Jan kent haar. $\rightsquigarrow$ 4 steps (excluding length 1)

Parsing methods and time

Parsing in dependency grammars:

n words $\rightsquigarrow$ $2n$ steps

Parsing in CYK:

n words$\rightsquigarrow$ $\frac{n^3-n}{6}+n$ steps (in the worst case)

Which is faster?

Parsing methods and time

Parsing in dependency grammars:

n words $\rightsquigarrow 2n$ steps

Parsing in CYK:

n words$\rightsquigarrow \frac{n^3-n}{6} + n$ steps (in the worst case)

Which is faster? Dependency parsing:

- 3 words $\rightsquigarrow 6$ steps
- 4 words $\rightsquigarrow 8$ steps
- 5 words $\rightsquigarrow 10$ steps
- 6 words $\rightsquigarrow 12$ steps

Parsing methods and time

Parsing in dependency grammars:

n words $\rightsquigarrow$ $2n$ steps

Parsing in CYK:

n words$\rightsquigarrow$ $\frac{n^3-n}{6} + n$ steps (in the worst case)

Which is faster?   CYK:

- 3 words $\rightsquigarrow$ 7 steps
- 4 words $\rightsquigarrow$ 14 steps
- 5 words $\rightsquigarrow$ 25 steps
- 6 words $\rightsquigarrow$ 41 steps

Parsing methods and time

Parsing in dependency grammars:

n words $\rightsquigarrow 2n$ steps

Parsing in CYK:

n words$\rightsquigarrow \frac{n^3-n}{6} + n$ steps (in the worst case)

Which is faster?

- Big-O notation: runtime as a proportion of input length (disregarding constants)
- Dependency parsing $= \mathcal{O}(n)$
  CYK $= \mathcal{O}(n^3)$

Parsing methods and time

Parsing in dependency grammars:

n words $\rightsquigarrow 2n$ steps

Parsing in CYK:

n words$\rightsquigarrow \frac{n^3-n}{6} + n$ steps (in the worst case)

Which is faster?

- Big-O notation: runtime as a proportion of input length (disregarding constants)
- Dependency parsing $= \mathcal{O}(n) \leftarrow$ much faster!
  CYK $= \mathcal{O}(n^3)$

Summary

- Dependency grammars: encode relations between words
- Projective and non-projective structures

  Some non-projective structures are part of language!
- Dependency parsing
  - Typically much faster–good for cases where we don't care about constituency
  - But: requires an oracle